Claims

1. Apparatus for processing data, comprising processing means and memory means for storing data and instructions for processing said data, wherein said memory means includes application instructions and data that define

an initialisation manager; and

a plurality of application modules;

each of said application modules includes a registration object for registering dependency of said module upon others of said application modules, to said initialisation manager;

each said application module further includes operational instructions defining operations of said module used by other modules; and

at least two of said application modules include initialisation instructions for initialising data affecting execution of said operational instructions;

said initialisation manager includes instructions for performing the steps of:

(a) processing said registered module dependencies to identify a dependency count for each module;

(b) generating an initialisation schedule by sorting the module order according to the number of dependencies; and

(c) calling said initialisation instructions in the order defined by said initialisation schedule.

2. Apparatus according to claim 1, wherein said registration object

for a module includes registration instructions that are called automatically as a result of loading the module.

3. Apparatus according to claim **1**, wherein said initialisation manager includes an initialisation list that records module dependencies.

4. Apparatus according to claim **1**, wherein said processing step (a) comprises steps of

(a1) creating a dependency array that defines non-transitive dependencies;

(a2) processing said dependency array to identify transitive dependencies; and

(a3) recording the total number of dependencies for each registered module.

5. Apparatus according to claim **1**, wherein said initialisation manager further includes instructions for initialising plug-in modules loaded after step (c), including the steps of:

(d) processing registered plug-in module dependencies to identify a dependency count for each plug-in module;

(e) generating an additional initialisation schedule by sorting the newly registered plug-in modules into order of number of dependencies;

(f) calling initialisation instructions in said plug-in modules in an order defined by said additional initialisation schedule; and

(g) extending the existing initialisation schedule by adding said additional initialisation schedule.

6. Apparatus according to claim **1**, wherein at least two of said application modules include finalisation instructions for finalising data affecting operational instructions.

7. Apparatus according to claim **6**, wherein said initialisation manager includes instructions for finalising modules by calling module finalisation instructions in the reverse of the order defined by the initialisation schedule.

8. Apparatus according to claim **1**, wherein one of said application modules includes the main application instructions, from which a call is made to invoke processing steps (a), (b) and (c), performed by the initialisation manager.

9. Apparatus according to claim **1**, wherein said memory means also includes automatic code generating instructions, for generating source code for an initialisation object in a module.

10. Apparatus according to claim **1**, wherein said application is an operating system.

11. A method of processing data in a processing system comprising processing means and memory means for storing data and instructions for processing said data, wherein said memory means includes application instructions and data that define

an initialisation manager; and

a plurality of application modules;

each of said application modules including a registration object for registering dependency of said module upon others of said application modules, to said initialisation manager;

each said application module further includes operational instructions defining operations of said module used by other modules; and

at least two of said application modules include initialisation instructions for initialising data affecting execution of said operational instructions;

said initialisation manager performing the steps of:

(a) processing said registered module dependencies to identify a dependency count for each module;

(b) generating an initialisation schedule by sorting the module order according to the number of dependencies; and

(c) calling said initialisation instructions in the order defined by said initialisation schedule.

12. A method according to claim **11**, wherein said registration object for a module includes registration instructions that are called automatically as a result of loading the module.

13. A method according to claim **11**, wherein said initialisation manager records said module dependencies in an initialisation list.

14. A method according to claim **11**, wherein said processing step (a) comprises steps of

(a1) creating a dependency array that defines non-transitive dependencies;

(a2) processing said dependency array to identify transitive dependencies; and

(a3) recording the total number of dependencies for each registered module.

15. A method according to claim **11**, wherein said initialisation manager further performs steps for initialising plug-in modules loaded after step (c), which include the steps of:

(d) processing registered plug-in module dependencies to identify a dependency count for each plug-in module;

(e) generating an additional initialisation schedule by sorting the newly registered plug-in modules into order of number of dependencies;

(f) calling initialisation instructions in said plug-in modules in an order defined by said additional initialisation schedule; and

(g) extending the existing initialisation schedule by adding said additional initialisation schedule.

16. A method according to claim **11**, including executing finalising instructions contained in at least two of said application modules.

17. A method according to claim **16**, wherein said initialisation manager calls said module finalising instructions in the reverse of the order defined by the initialisation schedule.

18. A method according to claim **11**, including making a call from the main application function to invoke processing steps (a), (b) and (c), prior to main application execution.

19. A method according to claim **11**, including executing automatic code generating instructions, thereby generating source code for an initialisation object in response to specified module dependencies.

20. A method according to claim **11**, wherein said application is an operating system.

21. A data structure defined upon a machine readable medium, comprising an initialisation manager and a plurality of application modules; wherein

each of said application modules includes a registration object for registering dependency of said module upon others of said application modules, to said initialisation manager;

each said application module further includes operational instructions defining operations of said module used by other modules; and

a plurality of said application modules include initialisation instructions for initialising data affecting execution of said operational instructions;

said initialisation manager includes instructions for performing the steps of:

(a) processing said registered module dependencies to identify a dependency count for each module;

(b) generating an initialisation schedule by sorting the module

order according to the number of dependencies; and

(c) calling said initialisation instructions in the order defined by said initialisation schedule.

22. A data structure according to claim **21**, wherein said registration object for a module includes registration instructions that are called automatically as a result of loading the module.

23. A data structure according to claim **21**, wherein said initialisation manager includes an initialisation list that records module dependencies.

24. A data structure according to claim **21**, wherein said processing step (a) comprises steps of

(a1) creating a dependency array that defines non-transitive dependencies;

(a2) processing said dependency array to identify transitive dependencies; and

(a3) recording the total number of dependencies for each registered module.

25. A data structure according to claim **21**, wherein said initialisation manager further includes instructions for initialising plug-in modules loaded after step (c), including the steps of:

(d) processing registered plugin module dependencies to identify a dependency count for each plugin module;

(e) generating an additional initialisation schedule by sorting the newly registered plugin modules into order of number of dependencies;

(f) calling initialisation instructions in said plugin modules in an order defined by said additional initialisation schedule; and

(g) extending the existing initialisation schedule by adding said additional initialisation schedule.

26. A data structure according to claim **21**, wherein at least two of said application modules include finalisation instructions for finalising data affecting operational instructions.

27. A data structure according to claim **26**, wherein said initialisation manager includes instructions for finalising modules by calling module finalisation instructions in the reverse of the order defined by the initialisation schedule.

28. A data structure according to claim **21**, wherein one of said application modules includes the main application instructions, from which a call is made to invoke processing steps (a), (b) and (c), performed by the initialisation manager.

29. A data structure according to claim **21**, wherein said memory means also includes automatic code generating instructions, for generating source code for an initialisation object in a module.

30. A data structure according to claim **21**, wherein said registered

application modules are part of an operating system.